Assured Design
Dr. John Goodenough
jbg@sei.cmu.edu

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213
Assured Design (SW-reliant system)

**Assured**: Having justified confidence that a software-reliant system has particular properties
Assured Design (SW-reliant system)

**Assured:** Having justified confidence that a software-reliant system has particular properties

- **Functional** Actions and outputs in response to inputs
- **Run-time** Reliability, security, safety, performance, etc.
- **Lifecycle** Modifiability, testability, etc.
Assured Design (SW-reliant system)

**Assured**: Having justified confidence that a software-reliant system has particular properties

- Functional Actions and outputs in response to inputs
- Run-time Reliability, security, safety, performance, etc.
- Lifecycle Modifiability, testability, etc.

**Design**: The structure (architecture) of a system — its constituents and their relationships
Assured Design (SW-reliant system)

Assured: Having justified confidence that a software-reliant system has particular properties

- Functional Actions and outputs in response to inputs
- Run-time Reliability, security, safety, performance, etc.
- Lifecycle Modifiability, testability, etc.

Design: The structure (architecture) of a system — its constituents and their relationships

Assured Design: Having justified confidence that a (software-reliant) system design has particular properties
Assured Design (SW-reliant system)

**Assured**: Having justified confidence that a software-reliant system has particular properties

- **Functional** Actions and outputs in response to inputs
- **Run-time** Reliability, security, safety, performance, etc.
- **Lifecycle** Modifiability, testability, etc.

**Design**: The structure (architecture) of a system — its constituents and their relationships

**Assured Design**: Having justified confidence that a (software-reliant) system design has particular properties

Design errors are costly and important
# Safety-Critical System Challenges

70% of faults introduced in requirements and architecture design
80% of faults discovered post unit test

<table>
<thead>
<tr>
<th>Where Faults are Introduced</th>
<th>Requirements Architecture Design</th>
<th>Code</th>
<th>Unit Test</th>
<th>Integration Test</th>
<th>Acceptance Test</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>70%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Where Faults are Found</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5%</td>
</tr>
<tr>
<td>16%</td>
</tr>
<tr>
<td>50.5%</td>
</tr>
<tr>
<td>9%</td>
</tr>
<tr>
<td>20.5%</td>
</tr>
</tbody>
</table>

80% of faults discovered post unit test

Nominal Cost Per Fault for Fault Removal

80% of faults discovered post unit test

Total System Dev. Cost
Boeing 777 $12B   F-35 $59B

Software as % of total system dev. cost
1997: 45% → 2010: 66% → 2024: 88%

Sources: Critical Code; NIST, NASA, INCOSE, and Aircraft Industry Studies
DoD Impact of Poor/Incorrect Design

Time to field and development cost
• Need for rework
• Extended T&E
Degraded sustainability

Reliance-21 C4I COI: Need to field new capabilities faster as threats and technologies change

Better designs are critical
Session Presentations

**Design**: Modeling and Analysis of Designs

- *Effective Reduction of Avoidable Complexity in Embedded SW*
- *Open Source AADL Workbench*
- *Extending AADL for Security Design Assurance of the Internet of Things*

**Implementation**: Vulnerability reduction; Exploit new HW

- *Increase Adoption of Secure Coding Standards*
- *Graph Algorithms on Future Architectures*
Session Presentations — Design

Modeling and Analysis of Designs
Session Presentations — Design

Modeling and Analysis of Designs

AADL (Architecture Analysis and Design Language):

- specifies a static representation of a system architecture
- can model logical flows, binding of software to hardware
- is strongly typed, allowing consistency checks with analysis tools
Session Presentations — Design

Modeling and Analysis of Designs

- *Effective Reduction of Avoidable Complexity in Embedded SW*
  - **DoD benefit**: Reduced T&E; Less rework → faster deployment; reduced cost
  - **SEI edge**: Arch. modeling expertise and tools (AADL)
Session Presentations — Design

Modeling and Analysis of Designs

• Effective Reduction of Avoidable Complexity in Embedded SW
  • DoD benefit: Reduced T&E; Less rework → faster deployment; reduced cost
  • SEI edge: Arch. modeling expertise and tools (AADL)

• Open Source AADL Workbench
  • DoD benefit: Less rework → faster deployment; reduced cost
  • SEI edge: AADL expertise; formal methods; real-time systems theory
Session Presentations — Design

Modeling and Analysis of Designs

• *Effective Reduction of Avoidable Complexity in Embedded SW*
  • DoD benefit: Reduced T&E; Less rework → faster deployment; reduced cost
  • SEI edge: Arch. modeling expertise and tools (AADL)

• *Open Source AADL Workbench*
  • DoD benefit: Less rework → faster deployment; reduced cost
  • SEI edge: AADL expertise; formal methods; real-time systems theory

• *Extending AADL for Security Design Assurance of the Internet of Things*
  • DoD benefit: Fewer operational vulnerabilities
  • SEI edge: Arch. modeling; model checking; SW security
Session Presentations

Design: Modeling and Analysis of Designs

- Effective Reduction of Avoidable Complexity in Embedded SW
- Open Source AADL Workbench
- Extending AADL for Security Design Assurance of the Internet of Things

Implementation: Vulnerability reduction; Exploit new HW

- Increase Adoption of Secure Coding Standards
- Graph Algorithms on Future Architectures
Session Presentations — Implementation

Prevent Vulnerabilities

• *Increase Adoption of Secure Coding Standards*

Secure Software Development Lifecycle
Session Presentations — Implementation

Prevent Vulnerabilities

- *Increase Adoption of Secure Coding Standards*

Secure Software Development Lifecycle
Session Presentations — Implementation

Prevent Vulnerabilities

- *Increase Adoption of Secure Coding Standards*
  - DoD benefit: Fewer operational vulnerabilities
  - SEI edge: CERT’s knowledge of vuls and access to code

Secure Software Development Lifecycle
Session Presentations — Implementation

Better Analysis Capabilities

- **Graph Algorithms on Future Architectures**
Session Presentations — Implementation

Better Analysis Capabilities

• **Graph Algorithms on Future Architectures**
  
  • **DoD benefit**: Faster exploitation of new HW architectures → improved security analysis capability
  
  • **SEI edge**: Knowledge of graph analysis algorithms and their use

![Graph Algorithms Examples](image1.png)

- **Community Detection**
- **Shortest Path**
- **Cost Minimization**
- **Max Flow**
- **Connected Components**
- **PageRank**
- **Centrality**
- **Clustering**

![Example Graphs](image2.png)

- APT Detection in Computer Networks, C3E, 2013
- United States Interstate Highway System
- Malware Distribution Networks
- Social Networks
- Revert graph showing editor conflict on the "Cyprus dispute" Wikipedia page, 2015
Summary

**Assured Design**: Advancing the state of the art and practice
- Modeling and analysis tools (AADL)
- Application in real-world contexts (JMR; SAVI)

**Coding Rules**: Reducing vulnerabilities in deployed systems

**Modifiability**: Ensuring graph analysis algorithms can readily exploit new HW architectures